

高级语言程序设计

第4章 基本程序设计技术

华中师范大学物理学院 李安邦

我要问同学们一些问题：

1. **通读了教材吗？** 通常教材应该读 3 遍：预习读 1 遍，听课时读 1 遍，期末复习时再读 1 遍。
2. **上课做了笔记吗？** 虽然老师总是把重要知识点贴在QQ群里，但那也不是足够的。重要的是学习者抓住课堂教学的重点和难点，根据自己的实际情况做适量的笔记。
(现在老师在QQ群里贴出重点和难点，是为了引导同学们学会做笔记)
3. **是否阅读了老师批阅后的源程序？** 有没有耐心地理解老师的批阅意见？有没有耐心地读一遍其他同学的源程序？是否会避免其他同学所犯的错误？
4. **有没有足量完成老师布置的上机编程练习？** 虽然老师只收取部分程序，但自己应该自觉地足量完成。
5. 有没有耐心地读了老师在QQ群里发的信息？有没有遵照老师的要求去做？

对勤奋好学的同学的寄语：

- 努力学习总是能够得到回报的。
- 努力学习的成果应该主动地向老师展示。例如：
如果你遵照老师的建议，做完了教材上所有的编程练习题，那么可以主动地把所编写的程序打包发送给老师，老师会欣然接受并做耐心的批阅，给你个人足够多的指导和帮助。

- 从前面的编程工作中可以体会到，虽然写程序时要处理许多琐碎细节，但这也是一件有趣的工作，是对人的智力挑战。
- 为了完成一个程序，首先要分析问题以寻找解决方案，为此需要聪明才智和想像力，各种相关领域的知识和技术都可能有用。
- 要把设计变成现实，变成可以运行的程序，既需要发挥智力，又需要有条有理地工作，还要非常细心。一个小错误就可以使程序无法编译或不能正确执行。
- 当然，高度精确性也是现代社会的需要，写程序的过程能提供许多有益的体验。

- 学习程序设计需要注意规律性的东西。
- 要注意前人的经验。
- 只有认真学习怎样写好小程序，弄清其中的基本道理，才能做出大程序。

三种流程模式的总结：

- 顺序模式最简单
- 选择模式：要确定判断条件及不同情况下的动作
- 开始的难点在实现重复执行的循环。重复执行比较复杂，牵涉问题多，是本章重点
- 本章还讨论：常用标准库函数和交互式程序设计中的输入处理

只有在程序设计实践中才能学好程序设计：

- 1、模仿好的范例；
- 2、自己动手动脑，反复实践从问题出发做出程序的整个过程；
- 3、多想想自己的程序做得怎么样，能不能做得更好。

目录

4.1 循环程序设计

4.2 常用标准库函数

4.3 交互式程序设计中的输入处理

4.4 程序设计实例

4.5 程序动态除错方法（二）

4.1 循环程序设计

- 写循环首先要发现循环。注意计算中的重复性动作，引进循环可能统一描述和处理。

重复动作的常见实例：

- 一批类似数据做同样加工处理
- 累积一批可按规律算出的数据（累加等）
- 反复从一个结果算出下一结果（递推）

- 若重复次数很多，就应该考虑用循环
- 如果重复次数无法确定，就必须用循环描述

- 从发现重复性动作到写好一个循环结构，还需考虑和解决许多具体问题：
 - ◆ 每次循环中需要做的处理工作；
 - ◆ 循环的控制……
 - ◆ 具体还包括使用语言里的哪种结构来实现循环。
- 本节将展示一批程序实例，介绍一些循环程序设计问题。
- 先分析问题，逐渐发掘完成程序的线索，最终完成能解决问题的程序。→ “从问题到程序”

读者需要理解：

- 程序不是教条。即使对一个典型问题，也没有需要死记硬背的标准解答。
- 只要不是极端简单的问题，总会有许多解决方法，可以写出许多在形式或实质上有差异的程序。
- 正确的程序也常有优劣之分。
- 读者需要学习写出优秀的程序。

4.1 循环程序设计

4.1.1 生成与检查

4.1.2 浮点误差

4.1.3 迭代和逼近

4.1.4 通项计算

4.1.5 循环中的几种变量

4.1.1 生成与检查

【例4-1】写程序，打印输出 1 到 200 之间的所有完全平方数，即那些是另一个数的平方的数。

通过分析可以发现，这样一个简单问题也有许多不同解法。

方法一：逐个检查，遇平方数打印。

重复做，每次检查一个数。循环框架：

```
for (n = 1; n <= 200; n++)
```

```
    if (n 是完全平方数) 打印 n;
```



没有直接判断手段，需要写一段程序填充

```
for (n = 1; n <= 200; n++)  
    if (n 是完全平方数) 打印 n;
```

可以顺序检查，是否存在某数，其平方恰为n。
这构成（循环内的）新循环，需要一个新循环变量。
k 可以从1开始，递增，直至 $k * k > n$:

```
    for (k = 1; k * k <= n; k++)  
        if (k * k == n) 打印 n;
```

```
int main() {  
    int n, k;  
    for (n = 1; n <= 200; ++n)  
        for (k = 1; k * k <= n; ++k)  
            if (k * k == n) //如果n是k的平方，即n是完全平方数  
                cout << n << " ";  
    cout << endl; //最后换一行  
    return 0;  
}
```

方法二：需要打印的一定是从1开始连续几个整数的平方，可从1开始打印到平方大于200为止。

```
for (n = 1; n * n <= 200; ++n)  
    cout << n * n << " "; /*注意打印什么*/
```

方法三：还可以考虑利用递推公式：

$$\alpha_1 = 1$$

$$\alpha_{n+1} = \alpha_n + 2n + 1$$

- 方法一的策略是：生成所有备选数据（1 到 200 的整数），检查排除不合格的。
- 编程策略中，生成与检查是解决问题的常用方法。



- 方法二和方法三是针对具体问题的特定方法。

补充例题：如果一个三位数的各位数字的立方和等于该数本身，则称该数为“水仙花数”。例如：
 $153 = 1^3 + 5^3 + 3^3$ 。打印出所有的“水仙花数”。

解题思路：做一个从100 到 999 的循环，依次分解出每个数的百位、十位和个位数字，如果满足条件则打印输出。

利用整数除法和取余。


```
int main() { //打印输出所有水仙花数
```

```
    int i, j, k, n;
```

```
    cout << "水仙花数: ";
```

```
    for (n = 100; n < 1000; n++) {
```

<--生成一批数据

```
        i = n / 100; //分解出百位
```

```
        j = n / 10 % 10; //分解出十位
```

```
        k = n % 10; //分解出个位
```

<--检查

```
        if ( n == i * i * i + j * j * j + k * k * k)
```

```
            cout << n << endl;
```

```
    }
```

```
    return 0;
```

```
}
```

一种错误写法如下：

```
int main() { //打印输出所有水仙花数
```

```
    int i, j, k, n;
```

```
    cout << "水仙花数：";
```

```
    i = n / 100;    //分解出百位
```

```
    j = n / 10 % 10; //分解出十位
```

```
    k = n % 10;    //分解出个位
```

```
    for (n=100; n<1000; n++) {
```

```
        if (n == i * i * i + j * j * j + k * k * k)
```

```
            cout << n << endl;
```

```
    }
```

```
    return 0;
```

```
}
```

以数学函数关系来看待
i, j, k 与 n 的关系！错。

计算机程序的本质，是
按顺序进行计算。

4.1.2 浮点误差

在计算机中用二进制浮点数表示小数，可能存在**浮点误差**，即在计算机中所保存的小数与实际的十进制小数之间有误差。

在通常情况下可能对计算结果影响不大，但是，**在某些特定情况下却对计算结果存在严重的影响**。

用一个含有利用循环语句进行求和的例题说明。

【例4-2】假定有一只乌龟决心去做环球旅行。出发时它躊躇满志，第一秒四脚飞奔，爬了1米。随着体力和毅力的下降，它第二秒钟爬了 $1/2$ 米，第三秒钟爬了 $1/3$ 米，第四秒钟爬了 $1/4$ 米，如此等等。现在问这只乌龟一小时能爬出多远？爬出20米需要多少时间？

显然，题目中要计算的是无穷级数和式 $\sum_{n=1}^{\infty} \frac{1}{n}$ 前面的有限一段之和。

由高等数学可知， $\sum_{n=1}^{\infty} \frac{1}{n} = \infty$ 。也就是说，只要乌龟坚持爬下去，它不但能完成环球旅行，也能爬到宇宙的尽头。

在题目中有两问，即分别要求出给定 n 的求和上限值时这个级数和式的值是多少，以及 n 的值为多大时该级数和式能达到给定的值。

我们想用这个例子研究一下浮点数的误差问题，并对用于保存累加求和值的变量分别设为 `double` 和 `float` 类型的情况做些比较。

```

int main() {
    int m, n;
    double x, dist;
    //计算乌龟 m 秒爬出的距离
    cout << "请输入乌龟爬行时间（以秒为单位）：";
    cin >> m;
    cout << "m = " << m << endl;
    dist = 0.0;
    for (n = 1; n <= m; ++n) {
        dist += 1.0/n; //注意不能写成 1/n
        if (n % 100000 == 0) //按一定的间隔输出中间值
            cout << "n= " << n << " dist=" << dist << endl;
    }
    n--; //循环结束时，n的值为 m+1，比实际所用值多1，
        所以要减1。
    cout << "乌龟在 " << n << " 秒钟之后爬行了 " << dist << " 米
        远。" << endl;
}

```

$$\sum_{n=1}^m \frac{1}{n}$$

```
//计算乌龟爬行 dist 米距离所需的时间
cout << "\n请输入待爬行的距离（以米为单位，例如20） :";
cin >> dist;
x = 0.0;
for (n = 1; x < dist; ++n) {
    x += 1.0/n;
    if (n % 100000 == 0)
        cout << "n= " << n << " x= " << setprecision(10) << x << endl;
}
cout << "乌龟爬行 " << dist << " 米需要 " << --n << " 秒钟。" << endl;
cout << "约为 " << n/3600 << " 小时，" << n/3600/24 << " 天。" << endl;
return 0;
}
```

教师测试练习

测试运行时，先输入一个时间（例如3600秒），程序会很快计算出：

乌龟在 3600 秒钟之后爬行了 8.76604 米远。

然后试着输入一个较短的距离（例如20米），程序会花费一段时间之后计算出：

乌龟爬行 20.0000000000 米需要 272400600 秒钟。

约为 75666 小时，3152 天。

这是很合理的计算结果。

程序中满足 $n \% 100000 == 0$ 才输出中间量的值。

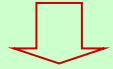
输入一个更大的时间和更长的距离，会怎么样？

显然，输入时间长度的时候，由于 `m` 是 `int` 类型，所以最大值只能是 `INT_MAX`，最终所计算得的距离还是一个合理的值。

如果输入一个更长的距离（例如 23），那么循环变量 `n` 就会溢出（出现负数！），结果没有意义。

计算机确实是一种数值范围有限的机器，它只能在有限的范围内进行工作。

double x, dist;



float x, dist;

- 把程序中的变量 x 和 dist 的类型改为 float，运行时会有什么样子呢？
- 结果是，大约在 2100000 秒之后，爬行距离最大达到 15.40368271 时就不再增长了。也就是说意味着，当 n 大于 2100000 时，对 x 的计算在 10 位精度下已经产生了明显的误差。

x += 1.0/n;

- 把变量 x 和 dist 的类型改为 long double 类型又会如何？

对浮点数类型的使用建议：

- float 的表示范围和精度不能满足许多常规计算的需要；
- 浮点计算通常都应该用 double 类型；
- long double 类型可能会影响程序效率，而且受到具体编译器的限制。



特殊情况中浮点误差积累可能更迅速。两个重要情况：

- 1) 将一批小的数加到很大的数上，会导致丢掉小的数的重要部分，甚至小数整个被丢掉（例中情况）
- 2) 两个值接近的数相减，可能导致精度大大下降

4.1.3 迭代和逼近

【例4-3】求 x 立方根的迭代公式（递推公式）是：

$$x_{n+1} = \frac{1}{3}(2x_n + x/x_n^2)$$

从键盘上输入 x 值，然后利用这个公式求 x 的立方根的近似值，要求达到精度 $|(x_{n+1} - x_n)/x_n| < 10^{-6}$ 。

- 无法确定循环次数，只能用循环。
- 只能按题目给出循环条件（计算方法的研究结果）
- 求新迭代值要用到 x 。判断终止需要先后两个近似值，必须用两个变量，这可看作是两个合作的迭代变量。

```
int main() {
    double x, x1, x2;
    cout << "input x: ";
    cin >> x;
```

```
    x2 = x;
    do {
        x1 = x2;
        x2 = (2.0 * x1 + x / (x1 * x1)) / 3.0;
        cout << x2 << endl;
    } while (fabs((x2 - x1) / x1) >= 1E-6);
```

```
x1 = x;       $x_{n+1} = \frac{1}{3}(2x_n + x/x_n^2)$ 
x2 = (2.0 * x1 + x / (x1 * x1)) / 3.0;
while (fabs((x2 - x1) / x1) >= 1E-6) {
    x1 = x2;
    x2 = (2.0 * x1 + x / (x1 * x1)) / 3.0;
    cout << x2 << endl;
```

```
}
    cout << "cubic root of x is : " << x2 << endl;
    return 0;
}
```

改用do ...while
可减少重复语句

精度不满足要求时做循环

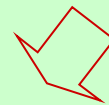
注意 x1 与 x2 的迭代变化!

多次运行，每次运行时**输入不同的数据进行测试它**。可以发现，在输入 27、-1000、3、-1 这样的数据时都能给出正确结果，但是在输入 **0** 时会得到输出结果为 **nan**。

$$x_{n+1} = \frac{1}{3}(2x_n + x/x_n^2)$$

```
int main() {  
    double x, x1, x2;  
    cout << "Please input x: ";  
    cin >> x;  
    x2 = x;  
    if (x == 0) {  
        cout << "cubic root of "<< x << " is : " << x2;  
        return 0; //程序结束，返回值为0  
    }  
}
```

添加对 0 的额外处理



.....

这个函数也很典型。人们研究了许多典型函数的计算方法，许多函数的计算都采用类似本函数计算方式的公式，其中需要通过一系列迭代计算，取得一系列逐渐逼近实际函数值的近似值。

实现这类计算的程序通常都具有上述函数的形式：采用几个互相协作的临时性变量，通过它们值的相互配合，最终算出所需要的函数值。

【例4-4】 Fibonacci（斐波纳契）数列：1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610

递推定义为：

$$F_1 = 1, F_2 = 1, \dots F_n = F_{n-1} + F_{n-2} \quad (n > 2)$$

请写程序，对于从键盘输入的介于 3 到 46 之间的正整数 n ，利用上述公式求出 F_n 。

程序中需要检查输入的 n 值是否介于 $[3, 46]$ 。

利用递推公式求 F_n ：从 F_1 和 F_2 出发，向前逐个推算，直至算出所需的 F_n 为止。

每次需要由前两项算出后一项，所以需要三个变量。开始时保存 F_1 、 F_2 和 F_3 ，然后逐步修改，使之在每个 $k > 2$ 时总对应着 F_{k-2} 、 F_{k-1} 和 F_k 。最后得到 F_n 。

```

int main() {
    int n;
    do {
        cout << "Please input n (between 3 and 46): ";
        cin >> n;
    } while (n < 3 || n > 46);
    long f1 = 1, f2 = 1, f3 = f1 + f2, k;
    cout << f1 << "\t" << f2 << "\t" << f3 << "\t";
    for (k = 4; k <= n; ++k) {
        f1 = f2;      f2 = f3;      f3 = f1 + f2;
        cout << f3 << "\t";
    }
    cout << "\nThe n item: " << f3 << endl;
    return 0;
}

```


递推 Fibonacci 数列各项：

Fibonacci数列： 1, 1, 2, 3, 5, 8, 13, 21, 34, 55

f1,f2,f3 初始： 1, 1, 2 (初值)

循环 1 (k=4):
↓ ↓
1, 2, 3

循环 2 (k=5):
↓ ↓
2, 3, 5

循环 3 (k=6):
↓ ↓
3, 5, 8

循环 4 (k=7):
↓ ↓
5, 8, 13

循环 5 (k=8):
↓ ↓
8, 13, 21

循环 6 (k=9):
↓ ↓
13, 21, 34

4.1.4 通项计算

【例4-5】 从键盘上输入一个 x 值，然后利用公式

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

求出 $\sin x$ 的近似值（单项绝对误差小于 $1\text{E-}6$ ），并与标准库中的 \sin 函数的计算结果进行比较。

方法：循环累加， n 趋向无穷的过程中项值趋于 0，而累加值趋向函数值。**需要用循环。**

保存累加和的变量 sum ，循环中求得项值用 t 保存。

sum = 0.0;

对 n 为 0 计算 t;

while (需要继续) {

sum = sum + t;

计算下一个 t;

}

循环结束条件：新项绝对值小于 10^{-6}

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

for (t = 0.0, i = 1; i <= 2 * n + 1; ++i)
t *= -(x * x / (2 * i) / (2 * i + 1));

有很多重复计算

找出前后项之间的关系：
$$t_n = -t_{n-1} \cdot \frac{x^2}{2n \cdot (2n+1)}$$

t = -t * x * x / (2*n) / (2*n + 1);

减少了重复计算

```

int main() {
    double x, sum, t;
    cout << "Please input x: ";
    cin >> x;
    sum = 0.0;
    t = x;
    int n = 0;
    while (t >= 1E-6 || t <= -1E-6) {
        sum = sum + t;
        n = n + 1;
        t = -t * x * x / (2*n) / (2*n + 1);
        cout << "n= " << n << " t= " << t << " sum=" << sum
        << endl;
    }
    cout << "my sin " << x << " is : " << sum << endl;
    cout << "standard sin " << x << " is : " << sin(x);
    return 0;
}

```

多次运行，每次输入不同的参数值进行测试：2, 5, 10, 20, 50, 100, 1000


可以发现结果与 x 值有关。

绝对值很大时耗时较长，误差很大。

```
int main() {  
    double x, sum, t;  
    cout << "Please input x: ";  
    cin >> x;  
    sum = 0.0;  
    t = x;  
    int n = 0;  
    while (t >= 1E-6 || t <= -1E-6) {  
        sum = sum + t;  
        n = n + 1;  
        t = -t * x * x / (2*n) / (2*n + 1);  
        cout << "n= " << n << " t= " << t << " sum=" << sum  
        << endl;  
    }  
    cout << "my sin " << x << " is : " << sum << endl;  
    cout << "standard sin " << x << " is : " << sin(x);  
    return 0;  
}
```

改进:

x = fmod(x, 2*3.1415926);



启示：

1、理解问题非常重要。发现了**项的递推性质能节省许多计算**（否则就要再写一个嵌套循环完成项的计算）。

2、**级数收敛性质**能帮人认识情况，改进计算方法。

总之，写程序时必须仔细考虑问题本身的性质。

课堂笔记

- 4.1.1 生成和检查：同一个问题有多种解法；常用“生成和检查”；还有一些针对具体问题的具体解法。
- 4.1.2 浮点误差 （1）浮点数通常都用 double 类型；极少用 float 和 long double。（2）循环时部分地输出中间量。
- 4.1.3 迭代和逼近：注意变量的迭代变化；合理地选择循环结构（while，do ... while 或 for）；选用典型的参数进行测试，完善程序。
- 4.1.4 通项计算：写循环时需要找出前后项之间的关系，以减少计算量；选用典型参数测试，完善程序。

4.1.5 循环中的几种变量

循环中常出现几类变量，注意这些有助于对循环的思考和分折。也是写循环程序的经验总结。

注意：分类不是绝对的，不同类别没有截然界限。

- 1) **循环控制变量(循环变量)**：循环前设初值，循环中递增/递减，达到/超过界限时循环结束。它们控制循环的进行/结束。for中常有这类变量。

```
for(n = 0; n < 10; ++n).....
```

```
for(n = 30; n >= 0; --n) ... ..
```

```
for(n = 2; n < 52; n += 4) .....
```

在循环体内**不要修改**循环变量

这种循环是固定次数的循环。这种循环可能展开。

2) **累积变量**：循环中常用 += 或 *= 等更新。初值通常用运算的单位元（累加的初值为 0；累乘的初值为 1 为初值）。循环结束时变量终值被作为循环计算结果。

3) **递推变量**：前两类变量的推广。几个协同工作的变量，每次由几个变量推出一个新值，其余依次更新。

对变量 x_1 、 x_2 、 x_3 ，循环体可能有序列：

$x_1 = x_2;$

$x_2 = x_3;$

$x_3 = \dots x_1 \dots x_2 \dots;$

写循环时要考虑和解决问题列表：

- 循环涉及到哪些变量，需引进哪些临时性变量？
- 循环如何开始？循环开始前给变量什么初值？循环中变量的值如何改变？
- 什么情况下继续（或终止）循环？
- 循环终止后如何得到所需结果？
- 用哪种结构实现循环，等等。

工作方式：分析问题，发掘线索，最终完成程序。

程序设计不是教条，典型问题也无标准答案。并非最简单的问题总有多种解决方法，往往各有长短。

“正确”程序常有优劣之分。

课后作业：

重新阅读教材，并上机练习 4-1 节所有 5 个示例程序，整理好笔记。

目录

4.1 循环程序设计

4.2 常用标准库函数

4.3 交互式程序设计中的输入处理

4.4 程序设计实例

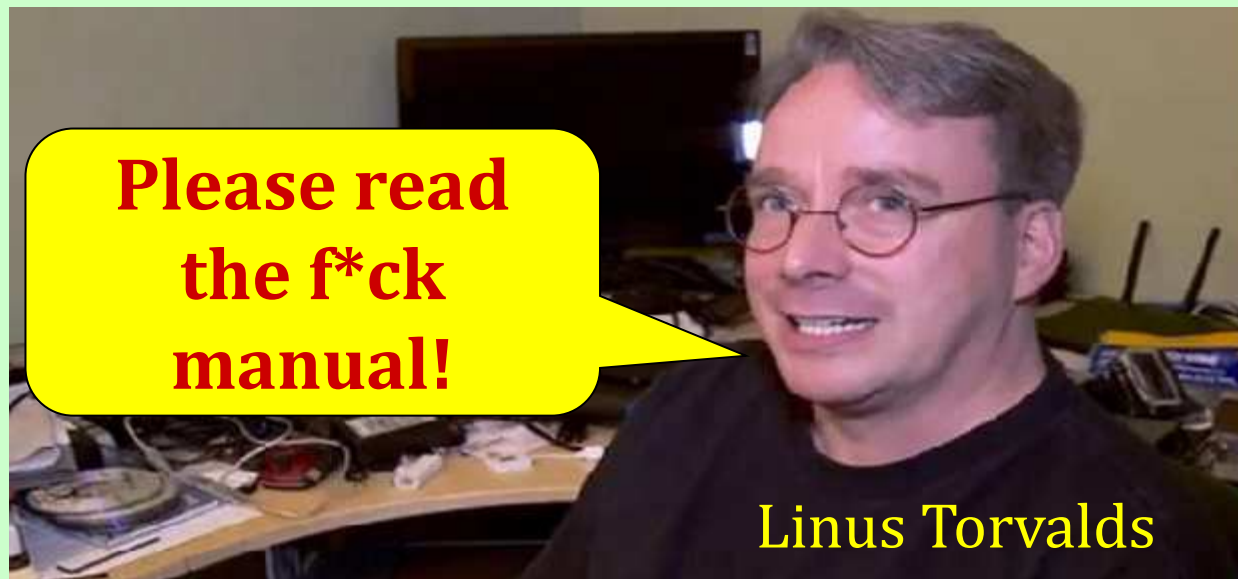
4.5 程序动态除错方法（二）

4.2 常用标准库函数

4.2.1 库函数

- 基本 C 和 C++ 语言很小，ANSI C 定义了标准库，其中提供最常用的与平台无关的功能（各种常用函数）。
- 每个符合标准的 C/C++ 系统都提供了标准库。
- 通常还提供一些扩充库，以便使用特定硬件/特定系统的功能：
- 扩充库不标准，使用扩充库的程序依赖于具体系统。
- 库函数实现常用计算，可按规定方式调用，不必自己实现/不必关心怎样实现。开发一次使所有用户受益。

- 标准库功能包括输入输出、文件操作、存储管理，其他如数学函数、数据转换函数等。有关介绍散布在各章。
- 下面介绍两组常用标准函数：程序计时，随机数。
- 对库函数的详细用法，可查阅系统联机帮助，或上网查阅有关手册、参考书籍。**需要耐心阅读并练习范例程序。**



国际上知名的编程高手林纳斯·托瓦兹通常都很耐心地回答别人提出的问题。但是有一次被人问了太多的入门级简单问题之后，忍不住脱口而出 “Please read the f*ck manual!”

4.2.2 程序计时

- 统计程序/程序片段的计算时间有助于理解程序性质。许多语言或系统都提供了内部计时功能。

- 使用程序计时功能，程序头部需要写：

`#include <ctime>`

- 做程序计时，通常需要用到：

1. 库函数 `clock()`。用于返回从“开启这个程序进程”时刻到“程序中调用`clock()`函数”时刻之间的CPU时钟计时单元（clock tick）数目。
2. 符号常量 `CLOCKS_PER_SEC`：表示一秒钟会有多少个时钟计时单元。

程序计时的用法

1. 在程序中需要计时的起始点和结束点分别调用 `clock()` 函数，得到这两次调用该函数的返回值（可能需要用变量保存。假设为 `t0` 和 `t1`）；
 2. 用这两次返回值之差除以符号常量 `CLOCKS_PER_SEC`：
$$(\text{double})(t1 - t0) / \text{CLOCKS_PER_SEC}$$

就得到计时起始点和结束点之间所经历的时间，所得结果以秒为单位。
- 每个系统有一个最小时间单位，小于这个时间单位的计时值都是 0，计时结果也是以这个时间单位步进增长的。所以得到的计时值只能作为参考。

【例4-6】 程序中经常要用到cout << 方法进行屏幕打印输出，那么单次执行cout << 屏幕打印输出一个整数要花费多少时间呢？多次执行 cout << 输出，统计求平均。

```
#include <iostream>
#include <ctime>
using namespace std;
int main() {
    int t0, t1;
    const int NUM = 50000;
    t0 = clock(); //计时起始点
    for (int i = 0; i < NUM; i++)
        cout << i << endl;
    t1 = clock(); //计时结束点
    cout << "Time elapsed: "
        << (double)(t1 - t0) / CLOCKS_PER_SEC << " s" << endl;
    cout << "per excution: "
        << 1000.0 * (t1 - t0) / CLOCKS_PER_SEC / NUM << " ms" << endl;
}
```

还要说到的是，在标准库中还有库函数 `time()`，用于取得从格林威治时间1970年1月1日零时整开始计时所经过的时间，以秒为单位。误差比较大。

用法举例：`t1= time(0);`

(这个函数常常用于其它用途)

4.2.3 随机数生成函数

- 应用有时需要带随机性的计算，为此需要生成随机数：
 - 程序调试：用数据做运行试验，随机数据非常合适。
 - 计算机模拟：模拟实际情况/过程，探索规律性。客观事物变化有随机性。
- 计算机只能生成“伪随机数”。
- 常用方法是定义某种递推关系，设法使序列中的数比较具有随机性。可以根据这种方法定义随机数生成函数（...）。

使用标准库的随机数功能，程序需包含头文件：

`#include <cstdlib>`


相关功能：

- 函数 srand: `void srand(unsigned seed)`

功能：用 seed 值设种子值。默认初始种子值是1。
根据种子值生成下一随机数并修改种子值。

常用技巧：用当前时间做种子：`srand(time(0));`

- 随机数生成函数：`int rand(void)`

功能：无参数，得到 0 和符号常量 `RAND_MAX` 间的随机整数（该数值与系统有关）。 

例：`k = rand();`

把随机数转换到一个自定义的范围：

- 转换到整数区间[min, max]：

$k = \text{rand}() \% (\text{max} + 1 - \text{min}) + \text{min};$

- 转换到 [0,1] 实数：

$\text{rand()} * 1.0 / \text{RAND_MAX}$

- 转换到实数区间[dmin, dmax]：

$x = \text{double}(\text{rand}()) / \text{RAND_MAX} * (\text{dmax} - \text{dmin})$
 $+ \text{dmin};$

【例4-7】打印输出在使用当前时间作为种子值之后产生的40个处于[0, RAND_MAX]范围内的整数随机数和40个处于区间 [-5.5, 10.5] 内的实数随机数，每行输出5个数字时换行。

```
#include <iostream>
#include <ctime>      // 使用 time() 函数所需的库文件
#include <cstdlib>     // 使用随机数函数所需的库文件
using namespace std;

int main() {
    int i, k;
    cout << "产生从 0 到 "<< RAND_MAX << " 的整数随机数: " << endl;
    srand(time(0));                                设置一次种子数，然后输出一批随机数
    for (i = 0; i < 40; i++) {
        k = rand();
        cout << k << (i % 5 == 4 ? '\n' : '\t');
    }
    cout << endl << endl;
```





```
double x, dmin = -5.5, dmax = 10.5;
cout << "从 " << dmin << " 到 " << dmax << " 的实数随机数: " << endl;
 srand(time(0));
for (i = 0; i < 40; i++) {
    x = double(rand()) / RAND_MAX * (dmax - dmin) + dmin;
    cout << fixed << x << (i % 5 == 4 ? '\n' : '\t');
}
return 0;
}
```

关于标准函数库 cstdlib 的附加说明

- 程序中只要包含头文件 cstdlib ，就可以使用C语言标准库（ standard library ）的功能：
- 五种类型： size_t、wchar_t、div_t、ldiv_t和lldiv_t；
- 宏，例如 EXIT_FAILURE、EXIT_SUCCESS、RAND_MAX 和 MB_CUR_MAX 等等；
- 常用函数，例如 rand()、srand()、malloc()、calloc()、realloc()、free()、system()、atoi()、atol()、exit() 等等。

目 录

4.1 循环程序设计

4.2 常用标准库函数

4.3 交互式程序设计中的输入处理

4.3.1 通过计数器控制循环输入

4.3.2 用结束标志控制的循环输入

4.3.3 输入函数的返回值及其作用

4.3.4 字符串流与文件流输入输出

4.3.5 字符输入输出与字符相关函数

4.4 程序设计实例

4.5 程序动态除错方法（二）

4.3 交互式程序设计中的输入处理

程序可以分为两类：

1. 启动后自己运行，可能提供一些输出之后结束；
2. 执行中需要不断与外界（用户）打交道，需要外界提供信息，这类程序称为交互式程序（有输入的程序都可以看作交互式程序）。

前文中的很多程序都是简单的交互式程序，只要求用户输入一两个数据即可。

如果程序中需要输入较多的数据，那么就需要考虑一些相应的处理方法才行。

4.3.1 通过计数器控制循环输入

如果**事先知道需要输入的数据项数**，就可以用**计数器**控制输入循环。

4.3.2 用结束标志控制的循环输入

假定写程序时**不知道**需要输入的数据的确切项数，就无法采用计数循环的简单方法。需要使用结束标志。

4.3.3 输入函数的返回值及其作用

当上面两种方式都不能用时，需要使用输入函数数据的返回值，判断是否继续输入。

4.3.4 字符串流与文件流输入输出

- 一、使用字符串类进行输入输出
- 二、使用流式文件进行输入输出

4.3.1 通过计数器控制循环输入

如果事先知道需要输入的数据项数，就可以用计数器控制输入循环。

【例4-8】 假设用户手头有一批 double 类型的数据，写一个程序，让用户先输入数据总项数，然后依次输入这些数据，最后求出所有这些数据的总和并输出。

这一程序中的输入操作可以通过一个计数的输入循环完成：

```
int main() {  
    int i, n;  
    double x, sum;  
    cout << "Please input number of data items: ";  
    cin >> n;  
    for (i = 1, sum = 0; i <= n; ++i) { //计数器  
        cout << i << " : ";  
        cin >> x; //输入  
        sum += x;  
    }  
    cout << "Sum= " << sum << endl;  
    return 0;  
}
```

有一些编程者偷懒，在输入之前不作任何提示，会导致程序运行时操作困难。

```
int main() {  
    int i, n;  
    double x, sum;  
  
    cin >> n;  
    for (i = 1, sum = 0; i <= n; ++i) {    //计数器  
  
        cin >> x;    //输入  
        sum += x;  
    }  
    cout << "Sum= " << sum << endl;  
    return 0;  
}
```

4.3.2 用结束标志控制的循环输入

假定写程序时不知道需要输入的数据的确切项数，就无法采用计数循环的简单方法。

一种方式是用一个特殊“结束标志”控制循环。该“结束标志”应是一个特殊输入值，具有与输入数据同样的类型，但又不是正常输入数据。

让程序在循环中不断检测得到的数据，一旦看到这个特殊数据，就知道用户要求结束了。

采用这种技术，循环结束条件就是写程序的人与使用者之间的一种约定，当输入满足约定时程序就结束。

【例4-9】 假定需要计算一批货物的总值，每次输入的是货物单价和数量。由于并不知道需要算的货物种类，因此无法使用计数循环。可以考虑用一种特殊“结束标志”通知程序所有数据都已输入完。例如用单价为 0 作为结束标志（因为在实践中不可能有货物的单价为0）：

```
int main() {  
    double price = 1, amount, sum = 0;  
    cout << "依次输入货物的单位和数量 (0 0 结束) " << endl;  
    while (price != 0) {           //用特殊标志结束循环  
        cout << "输入下一项数据 (单价 数量) :";  
        cin >> price >> amount;  
        sum += price * amount;  
    }  
    cout << "Total price: " << sum << endl;  
    return 0;  
}
```


4.3.3 输入函数的返回值及其作用

- 在例4-9中，事先并不知道需要通过循环输入的数据的确切项数，但是所输入的货物单价总应该是正数，因此可以用某个特殊值作为输入结束标志。
- 如果在更一般的情况下，所输入的数据涵盖了可能的所有取值范围，那就无法选用某个特殊值作为输入结束标志了。在这种情况下，我们需要通过**输入函数的返回值**传递信息，以控制循环的结束。
- 在使用 `cin >>` 进行流式输入时，实际上**提取运算符 “>>” 会返回一个值**。在程序运行时如果用户输入了一个**合法**的数据给后面的变量，那么 “>>” 就会返回获得一个**非零值**。当用户输入一个非法的数据时，就会返回**零值**。
- 这个返回值可以用在程序中用于判断所输入的数据是否合法，从而控制是否进行其它操作。

【例4-10】 请输入一系列 double 类型的数据（总项数事先未知），对数据项数计数，并求出所有数据的累加和。
使用cin >> 输入数据时的返回值，写出程序如下：

```
int main() {  
    int n = 0;  
    double x, sum=0;  
    cout << "input x:";  
    while( (cin>>x) ) { //使用 “>>” 的返回值作循环控制  
        n++;  
        sum += x;  
        cout << "input x: ";  
    }  
    cout << "n= " << n << endl;  
    cout << "sum= " << sum << endl;  
    return 0;  
}
```

输入非数值型数据时
（输入字母，或按Ctrl+Z），
输入循环就会中止。

上面这个输入方法也有别的用途。

在交互式程序的实际运行中，**正常用户也可能会出现操作失误**，这时需要程序能检测到输入错误，允许用户重新输入。

另一方面，也有可能**某些用户会恶意地输入错误数据**，程序应该检测到输入错误，强制用户重新输入合法数据。

【例4-11】在例4-7中，在运行时有两条语句（即“`cin >> n;`”和“`cin >> x;`”）要求用户输入数据。如果我们考虑到**要防止用户输入错误数据**（例如在要求输入数值型数据时却输入字符数据），可以在**每次输入时都进行检测，允许用户输入每项数据时允许最多出错三次**，可以改写成这样：

```

int main() {
    int i, n;
    double x, sum;
    int ierr = 0, ERRNUM = 3;
    cout << "请输入数据项数: ";

    while (!(cin >> n) || n <= 0) { //获得输入，并处理可能的出错情形
        ierr++;
        if (ierr <= ERRNUM) { //输入出错次数低于最大允许次数
            cin.clear(); //清除错误标记
            cin.sync(); //清空缓冲区
            cout << "输入出错 " << ierr << " 次。请重新输入数据项数: ";
        } else { //输入出错次数超过最大允许值
            cout << "\n致命错误：输入出错超过 " << ERRNUM << "次！ \n";
            exit(1); //结束程序
        }
    }
    cin.sync(); //成功获得输入数据之后，也要清空缓冲区
    cout << "n = " << n << endl;
}

```

```

for (i = 1, sum = 0; i <= n; ++i) {
    cout << i << " : ";
    ierr = 0;
    while (!(cin >> x)) { //获得输入，并处理可能的出错情形
        ierr++;
        if (ierr <= ERRNUM) { //输入出错次数低于最大允许次数
            cin.clear(); //清除错误标记
            cin.sync(); //清空缓冲区
            cout << "输入出错 " << ierr << " 次。请重新输入数据: ";
        } else { //输入出错次数达到最大允许次数
            cout << "\n致命错误：输入出错超过 " << ERRNUM << " 次! \n";
            exit(1); //结束程序
        }
    }
    sum += x;
}
cout << "Sum = " << sum << endl;
return 0;
}

```

对计算机的“输入”行为详细解释。

- 可以把标准输入（通常是键盘）看成一个绵延不断的字符序列（字符流），键入回车键时就把前面所键入的字符序列存入到系统的缓冲区。



- 程序里调用输入函数，就是想用掉该序列最前面的一个或几个字符。例如，用 `cin >>` 输入数值型数据时就会用掉序列前面的几个字符（如果它们符合转换要求）。
- 输入序列中的字符用掉一个就少一个，未用的字符则仍然留在序列中。如果 `cin >>` 工作中某个指定转换失败，就会返回一个错误标记，而输入流中的字符序列仍然保持在读入前的状态（读入失败的字符没有被用掉）。



```
if (ierr <= ERRNUM) { //输入出错次数低于最大允许次数
    cin.clear(); //清除错误标记
    cin.sync(); //清空缓冲区
    cout << "输入出错 " << ierr << " 次。请重新输入数据项数: ";
}
```

- 对每次输入都进行检测，如果出错次数少于3次，则用 **cin.clear()** 清除错误标记，并用 **cin.sync()** 清空缓冲区（这两个操作必须配合一起使用），然后提示用户重新输入。
- 当某项数据输入出错达到 3 次时，则用 **exit** 函数退出程序。—— **exit**函数通常是用在程序中用来终结程序用的，使用后程序自动结束，返回到操作系统。通常用**exit(0)**表示程序正常退出，**exit(1)**或**exit(-1)**表示程序异常退出。
- 如果对于所输入数据的值还有进一步要求，也可以写在条件中。

4.3.4 字符串流与文件流输入输出

- 如果多次运行程序，可能需要**多次重复输入数据**。能否只输入一次数据，以后重复使用数据呢？
- 在程序中可以用流式方法输入，来源并非限定为只能用标准键盘，而是可以用其它来源。
- C++ 中提供了更为简单的办法：用**字符串或文件**进行输入输出。

- C++ 的标准库中定义了一个称为“字符串类(string class)”的数据类型，这种类型的变量特别适合用于存储一系列的字符。在一个string类型的变量中可以存储任意多个字符。
- 字符串类的变量的定义方式与其它类型变量类似，而且定义时可以进行初始化赋值。例如：

```
string str1; //定义字符串str1（不作初始化）  
string str2 = "Hello, World!"; //定义并初始化
```
- 这种变量也可以用 cin >> 方式进行输入（输入时遇到空格、制表符或回车符为止），用 cout << 方式进行输出。例如：

```
cin >> str1;  
cout << str1 << endl << str2 << endl;
```

- 要想把字符串类的变量用作输入输出，需要把它们**绑定为输入输出字符串流（string stream）**。C++引入了相关的类型以支持相应操作：

istringstream类——输入；ostringstream类——输出；
stringstream类——输入输出。

- 程序中需要包含头文件 sstream，：

```
#include <sstream>
```

- 把一个字符串类的变量绑定为字符串输入流：

```
istringstream inss(str);
```

字符串输入流 已赋初值的字符串类的变量

- inss就成为了一个输入流，可以类似于cin一样用提取运算符“>>”进行输入操作。例如：

```
inss >> n; //从inss中读取一个数据并赋给整型变量n
```

【例4-12】对于例4-8，在程序中**改用一个string类型的变量存储一些示例数据用作输入源**，实现该例的功能：先读入数据总项数，然后依次读入数据，最后求出所有这些数据的总和并输出。

```
#include <iostream>
#include <sstream>
using namespace std;
```

```
int main() {
    int i, n;
    double x, sum;
    string str="8 1.2 3.5 6.4 4.7 8.9 10.5 5.8 9.4"; //示例数据
    istringstream inss(str); //定义字符串输入流变量并绑定到字符串变量
    inss >> n;
    cout << "number of data items: " << n << endl;
    for (i = 1, sum = 0; i <= n; ++i) {
        inss >> x;    cout << i << ": " << x << endl;    sum += x;
    }
    cout << "Sum= " << sum << endl;
    return 0;
}
```

根据后续语句的功能而设定初始值！

第一个数字为数据个数，后续为数据。



```
#include <iostream>
#include <sstream>
using namespace std;
```

```
int main() {
    int i, n;
    double x, sum;
    string str="8 1.2 3.5 6.4 4.7 8.9 10.5 5.8 9.4"; //示例数据
    istringstream inss(str);
    inss >> n;
    cout << "number of data is: " << n << endl;
    for (i = 1, sum = 0; i <= n; i++) {
        inss >> x;
        sum += x;
    }
    cout << "Sum= " << sum << endl;
    return 0;
}
```

注意：计算机只能按顺序从前往后地依次读取数据！
有个别同学认为，字符串里的数据可以随便排列，计算机总是会聪明地先读取出数据总数，然后再依次逐个读取数据。错了！计算机是蠢笨的！编程者必须仔细地排列好数据，并仔细地安排计算机依次读取数据。

二、使用流式文件进行输入输出

- 把待输入的数据一次性写在一个字符串中，不再需要每次都重复输入数据，减轻了调试工作的麻烦。这是它的优点。但不足之处在于，数据被写在程序内部，只有编程者能修改数据，不懂得编程的用户就没法使用这个程序了。
- 考虑到这一点，“**数据与程序分离**”就变得很有必要了。为此，有必要**使用文件作为输入源**。

（当然，编程者应当对计算机中的文件有基本的了解……）

文件的基本概念

1、文件和文件名

- **文件 (File)** 是操作系统中用于组织和存储各种信息的基本单位。
- 文件名包含**主名**和**扩展名**两部分，用“.”分隔。
- 文件的命名需要满足一些强制性条件：
- 使用英文和汉字，可以有空格和“.”（最后一个“.”被认为是文件主名与扩展名的分隔符）
- **不能有 ? * / \ > < : 等特殊符号。**
- 同一文件夹中的文件名或文件夹名不能同名。

2、文件扩展名

- 通常用合适的扩展名说明文件类型。
- 例如：C/C++ 源程序的扩展名为“.c”或“.cpp”，可执行程序的扩展名为“**exe**”。
- 约定俗成的扩展名：文本文件(**txt**)、图象文件(bmp, jpg, gif, png, tif)、多媒体文件(wav, mp3, wma, avi, mpg, rmvb, wmv, swf, flv)、Word文档(doc, docx)，电子表格(xls, xlsx)、演示文档(ppt, pptx)
- 在Windows系统中，文件的扩展名可能默认为隐藏，需要设置才能显示：点击菜单“工具”->“文件夹选项”，去掉勾选“隐藏已知文件类型的扩展名”



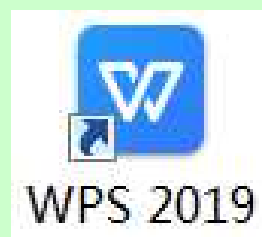
隐藏已知文件类型的扩展名

● 如何识别文件类型?

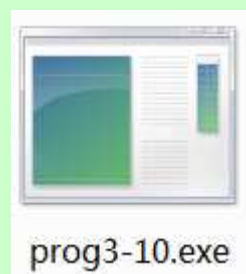
1、根据图标：可执行程序具有自己独特的图标（嵌在文件内部）。



2、可执行程序的快捷方式会显示为相同图标（左下角有小箭头）



3、其它同一类型的文件具有同一种图标。



文件目录

为了便于对文件进行存取和管理，计算机系统建立文件的索引——文件名与文件物理位置之间的映射关系，这称为**文件目录**（File Directory）。

目录在操作系统中也是特殊文件，命名规则和文件基本相同。

在图形化用户界面（例如Windows）中，目录被称为**文件夹**（Folder）——与目录是等价的。

操作系统采用**目录树**或称为**树形文件系统**的结构形式来组织系统中的所有文件。

1、树形文件目录结构

树形文件目录结构是一个有多层分布的目录及各级目录中的文件组成的结构形式，从磁盘开始，越向下级分枝越多，形成一棵倒长的树。

最上层的目录叫**根目录**。Windows 中，每一个驱动器的根目录用驱动器盘符加 **** 表示：C:\ D:\ E:\

2、当前目录、父目录与子目录

- 当前目录：用户正在工作的目录，用名字或 “.” 表示
- 父目录：某一目录的上级目录，用名字或 “..” 表示
- 子目录：某一目录的下级目录。

在资源管理器的“文件夹”窗格可以看到

3、目录路径：指明从根目录（或当前目录）开始到文件所在目录下所经历的各级目录名组成的序列。

- 目录之间用 \ 隔开（网址是用 / 隔开）
- 可以采用两种方式表示：绝对路径和相对路径。
- ◆ 绝对路径：从根目录开始，直到该文件所在的目录为止的目录路径。例如：

`c:\windows\system32\cmd.exe`

- ◆ 相对路径：指从当前目录开始到该文件所在目录为止的目录路径。例如，（如果当前是 C:\windows）
`system32\cmd.exe`

在资源管理器的“地址栏”中可以看到

- 在C++程序中，要使用计算机中的文件作为读取数据的来源或输出数据的目标，就需要把相应的文件绑定为文件输入输出流。
- C++引入了相关的类型：ifstream类——文件流输入；ofstream类——文件流输出；fstream类——输入输出。
- 程序中必须包含头文件 <fstream>：#include <fstream>
- 定义一个文件输入流的方法如下：
 ifstream infile; //infile为文件输入流的名称
- 要想通过一个文件输入流对象打开一个纯文本文件，需要使用它的成员函数open()：
 infile.open("data.txt") // “data.txt”为纯文本文件
- 也可以直接定义文件输入流并同时绑定纯文本文件：
 ifstream infile("data.txt");
- 然后就可以像使用 cin 一样从文件输入流读取数据：
 infile >> n; infile >> x;

只含人类可读字符的文件

- 类似的，定义一个文件输出流、并绑定纯文本文件：

```
ofstream outfile("output.txt");
```

文件输出流 可供输出数据的文件

- 然后就可以把 outfile 类似于 cout 一样使用插入运算符“<<”进行输出操作。例如：

```
outfile << n; //把整型变量n的值输出到outfile:
```

- 打开的文件使用完成后一定要关闭，fstream提供了成员函数close()来完成此操作：

```
infile.close();
```

```
outfile.close();
```

【例4-13】对于例4-10，在程序中**改用一个文本文件存储一些示例数据用作输入源**，实现该例的功能：读入一系列double 类型的数据（总项数事先未知），对数据项数计数，并求出所有数据的累加和。

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    int n = 0;
    double x, sum=0;
    ifstream infile("data.txt");
    if (!infile) { //如果输入文件流创建出错
        cout << "ERROR: can't open input file." << endl;
        exit(1);
    }
```

```
cout << "read data from input file:" << endl;
while( (infile >> x) ) {
    n++;          sum += x;    cout << x << endl;
}
infile.close();
```

```
ofstream outfile("output.txt");
if (!outfile) { //如果输出文件流创建出错
    cout << "ERROR: can't open output file." << endl;
    exit(1);
}
outfile << "n= " << n << endl;
outfile << "Sum= " << sum << endl;
outfile.close();
cout << "results saved in file output.txt" << endl;
return 0;
}
```

4.3.5 字符输入输出与字符相关函数

上面讨论的都是数值型数据的输入。
接下来讨论字符型数据的输入和输出。

一、从标准设备输入和输出字符

字符型数据也可以用cin >> 输入，用cout << 方式输出：

```
char ch;
```

```
cin >> ch;           //不能接收空格键和回车键。
```

```
cout << ch;
```


- 把键盘上输入的每个字符（包括空格和回车键）都作为一个输入字符给字符型变量时的办法：
 1. 使用标准库函数中专门用于输入字符 `getchar` 函数；
 2. 使用标准输入流对象 `cin` 的成员函数 `get`，它也专门用于输入字符的函数，在使用时写成 `cin.get` 的形式。

这两者的功能基本上是等价的，用户任选使用。

- 输入字符：`getchar` 和 `cin.get` 都是没有参数的函数（简称“无参函数”），它们都能从标准输入读一个字符，返回该字符的编码值。其类型特征描述为：

```
int getchar(void)
```

```
int cin.get(void)
```



是 `int` 类型而非 `char` 类型

- 标准库定义了一个符号常量 **EOF**（意为“End Of File”，文件结束），`getchar` 和 `cin.get` 在读字符时如果遇到文件结束（或输入结束），就返回EOF的值，说明已经没有输入了。在实际运行程序时，通常用组合键 `Ctrl+Z` 输入EOF。
- 一般系统里把 EOF 的值定义为 -1。
- EOF 值必须与任何字符的编码值都不同（否则就会产生混乱）。所以 C 和 C++ 标准库里把 `getchar` 和 `cin.get` 的返回值类型定义为 `int`（而不是 `char`）。
- 同样，调用 `getchar` 和 `cin.get` 时也应当用 `int` 类型的变量接收其返回值，这样才能保证 EOF 信息不丢失，保证文件结束判断的正确性。

- getchar 和 cin.get 的典型使用:

```
int ch;
```

```
ch = getchar();      //空括号不可少
```

```
ch = cin.get();
```

- 这两个语句都能从标准输入读一个字符，并把字符的编码赋给变量 ch。
- 默认情况下标准输入连到键盘，因此，当程序执行到这个 getchar 或 cin.get 调用时，该函数将从键盘取字符。如果没有已经键入的字符，程序就会等待，直到人通过键盘输入字符，并按过回车键之后，函数 getchar 或 cin.get 才能得到结果，语句完成后程序继续运行下去。

- 输出字符：标准库中的 **putchar** 和 `cout` 的成员函数 `put`（写作 **`cout.put`**）专门用于输出字符。它们的功能基本上等价，用户在实际应用中可以任选一种方式。例如：

```
putchar('O');
```

```
cout.put('K');
```

- 在运行时，两个字符‘O’和‘K’将被依次送到标准输出（通常是计算机显示器），因此，执行这两个语句的效果是在计算机屏幕上显示出“OK”。
- 通常可以用 `cout <<` 的方式输出，所以很少使用 `putchar` 和 `cout.put`。

在程序中如果需要以循环方式输入一系列字符，可以按照前文所说的两种方法来处理：在事先已知输入字符的个数时，用固定次数的循环；在事先未知待输入字符的个数时，可用特殊标志来控制循环，也可以用函数的返回值来控制（通常用EOF来结束）。

【例4-14】输入一系列字符，然后逐个输出，并输出字符对应的 ASCII 值，最后输出字符个数。请考虑是以回车符结束或者允许接受回车符。

借鉴前文讲到的“以输入函数的返回值”方法，写出程序：

```
#include <iostream>
using namespace std;
int main () {
    int ch, n = 0;
    cout<< "please input some chars: " <<endl;
    while ((ch = cin.get()) != '\n'){ //以回车符为结束标志
//while ((ch = cin.get()) != EOF ){ //以EOF为结束标志
        cout << (char)ch << "\t" << ch << endl;
        ++n;
    }
    cout << "Totally get " << n << " chars."<<endl;
    return 0;
}
```

- 这个程序执行时立即进入等待状态，等待输入字符。用户由键盘输入一个字符并按回车键后，程序将把输入的字符及其相应的ASCII值输出到屏幕。
- 当程序中**以回车符为结束标志时**，所输入的回车符并不被认为是输入字符（也不会输出其相应的ASCII值），程序这样就正常结束了。
- 当程序中**以EOF为结束标志时**，所输入的回车符还被认为是输入字符（并输出其相应的ASCII值），用户还可以继续上述操作，直到用户在新的一行输入中按下Ctrl+Z并键入回车键为止（如果紧接着普通字符后面按下Ctrl+Z，会被认为是输入一个ASCII值为26的含义为“替代”的特殊字符，并不算输入EOF标志）。
- 在此程序中的 `cout <<` 语句中以两种方式输出变量ch的值，第一种是 `(char)ch`，是先进行强制类型转换，把ch 转换为char类型再输出，所输出的是一个字符，另一种方式是直接输出ch的值，所输出的是该字符所对应的ASCII码。

二、从字符流和文件流输入输出字符

- 当然也可以从字符串流和文件流输入字符，或者向字符串流和文件流输出字符。方法与“4.3.4 字符串流与文件流输入输出”中是一样的，即用 >> 进行输入（类似于 cin >>），用 << 进行输出。
- 除此之外，字符串流和文件流也像 cin 一样具有成员函数 get。所以它们都可以类似于 cin.get 的方式输入一个字符。例如：

```
string str = "Hello,World";
```

```
char ch = str.get(); //从字符串类变量读取字符
```

或

```
ifstream infile("plain.txt");
```

```
char ch = infile.get(); //从文件输入流中读取字符
```

- 在下一节的实例程序中介绍这些用法。

三、与字符相关的标准库函数

- 在编写处理字符的程序时，可能需要对字符的性质进行判断，例如，要判断一个字符 `ch` 是否为字母或数字字符，如果根据字符的 ASCII 值进行判断：

```
(ch >= 48 && ch <= 57) || (ch >= 65 && ch <= 90) || (ch >= 97 && ch <= 122))
```

- 标准库中提供了一批处理字符的函数。按照C++标准，这些函数被包括在 `iostream` 库中（而按照C语言标准，需包含头文件 `ctype.h`）。
- 标准库中的各种字符分类函数都很简单，它们对满足条件的字符返回非 0 值，否则返回 0 值。→ 谓词函数。
- 它们的返回值被作为逻辑值使用，通常用来控制程序流程，或放在条件表达式的控制部分。出于可读性起见，谓词函数的名称通常以英语单词 “is” 开头。

字符分类

<code>isalpha(ch)</code>	ch是否字母字符
<code>isdigit(ch)</code>	ch是否数字字符
<code>isalnum(ch)</code>	ch是否字母或数字字符
<code>isspace(ch)</code>	ch是否空格、制表符、换行符
<code>isupper(ch)</code>	ch是否大写字母
<code>islower(ch)</code>	ch是否小写字母
<code>iscntrl(ch)</code>	ch是否控制字符
<code>isprint(ch)</code>	ch是否可打印字符，包括空格
<code>isgraph(ch)</code>	ch是否可打印字符，不包括空格
<code>isxdigit(ch)</code>	ch是否十六进制数字字符
<code>ispunct(ch)</code>	ch是否标点符号

`int tolower(int ch)` 转为对应小写字母

`int toupper(int ch)` 转为对应大写字母

【例4-15】 用户从键盘上输入一系列字符（以键入EOF结束），程序中分类统计所输入字符中的数字字符、小写字母和大写字母的个数。

```
int main() {  
    int ch, cd = 0, cu = 0, cl = 0; //digit, upper, lower  
    cout << "Please input some characters: " << endl;  
    while ((ch = cin.get()) != EOF) {  
        if (isdigit(ch)) ++cd;  
        if (isupper(ch)) ++cu;  
        if (islower(ch)) ++cl;  
    }  
    cout << "digits: " << cd << endl;  
    cout << "uppers: " << cu << endl;  
    cout << "lowers: " << cl << endl;  
    return 0;  
}
```

目录

4.1 循环程序设计

4.2 常用标准库函数

4.3 交互式程序设计中的输入处理

4.4 程序设计实例

4.4.1 编程实例1：一个简单猜数游戏

4.4.2 编程实例2：一个简单计算器

4.4.3 编程实例3：文件中的单词计数

*4.4.4 编程实例4：图形界面程序

4.4.1 编程实例1：一个简单猜数游戏

【例4-16】 写一个简单交互式游戏。程序自动生成一个位于某范围里的随机数，要求用户猜这个数。用户输入一个数后，程序有三种应答：too big, too small, you win。重复此游戏，直到用户希望结束为止。

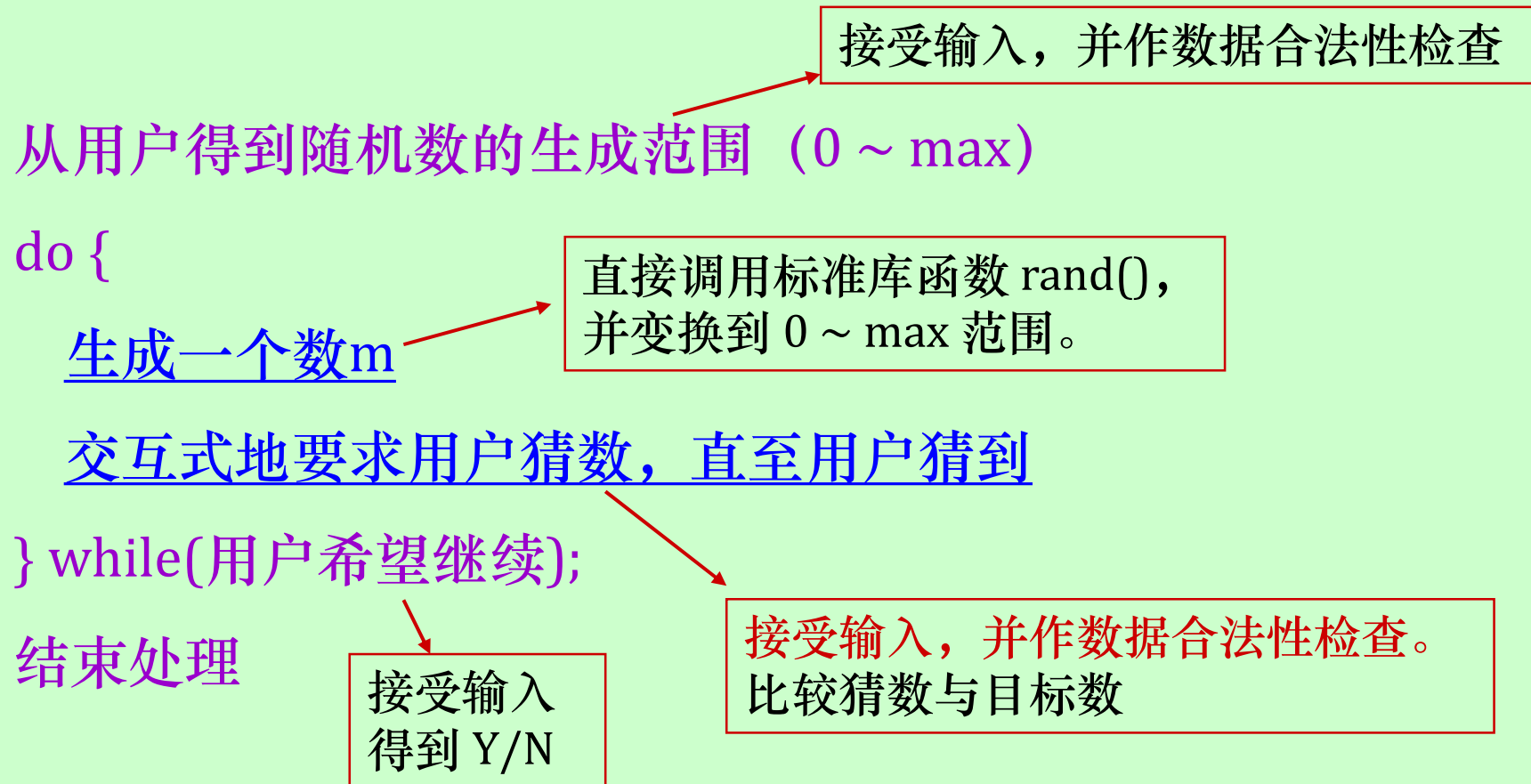
【分析】

可以用**随机数生成器**产生随机数。

在程序开始时要求用户提供一个范围，然后进入游戏循环。每次用户猜出一个数后询问是否继续。

这个程序的主要部分是一系列交互式的输入和输出。

整个程序的工作流程的基本设计：



```
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
    int max, m, guess, ch;
    const int ERRORNUM = 5; //允许的最大出错次数
    int err1 = 0, err2 = 0; //err1记录输入出错次数; err2记录猜数出错次数
    cout << "Number-Guessing Game" << endl;
    //设定最大值
    cout << "Choose a range [0, max]. Input max: ";
    for (err1 = 0; !(cin >> max) && err1 < ERRORNUM-1; err1++) {
        cin.clear(); //清除错误标记
        cin.sync(); //清空缓冲区
        cout << "Input again: ";
    }
}
```

```
if (err1 == ERRORNUM) { //输入出错次数太多
    cout << "Too many input errors! exit!"<<endl;
    exit(1);
}
cout << "max = " << max << endl;

srand(time(0)); //设定随机数种子
do { //程序主循环
    m = rand() % (max + 1);
    //产生新的待猜数字(注意取模的数为max+1)
    cout << endl << "A new rand number generated."<< endl;
    err2 = 0;
```



```
while (1) { //猜数循环
    err1 = 0;
    cout <<"Your guess: ";
    while(!(cin >> guess) && guess > max && err1 < ERRORNUM ) {
        //获得用户输入，并处理可能的出错情形
        err1++;
        cout << "Wrong. Need a number in 0~ "<< max << "."<<endl;
        cin.clear();
        cin.sync();
        cout <<"Your guess: ";
    }
    if (err1 == ERRORNUM) { //输入出错次数太多
        cout << "Too many input errors! exit!"<<endl;
        exit(1);
    }
}
```

```
//评价猜测结果
```

```
    if (guess > m) {
```

```
        cout << "Too big!" << endl;
```

```
        err2++;
```

```
    } else if (guess < m) {
```

```
        cout << "Too small!" << endl;
```

```
        err2++;
```

```
    } else { // guess ==m
```

```
        cout << "Congratulation! You win!" << endl << endl;
```

```
        break;
```

```
    }
```

```
    if (err2 == ERRORNUM ) {
```

```
        cout << "Too many errors. Stop!" << endl;
```

```
        break; /* 猜数时出错次数太多 */
```

```
    }
```

```
} //猜数循环结束
```

```
cin.clear();  
cin.sync();  
//继续游戏吗  
cout << "Next game? (y/n): ";  
while ((ch = toupper(cin.get())) != 'Y' && ch != 'N')  
    ; //空循环体  
cin.clear();  
cin.sync();  
} while (ch != 'N'); //程序主循环结束  
cout << "Game over. Thanks for playing." << endl;  
return 0;  
}
```

由于考虑了处理在输入环节中用户可能出现的错误，**整个程序是很健壮的**，即使用户故意多次输入错误，程序中也能正常地处理（多次提示用户正确输入，出错次数太多时就正常地中止程序，必要时清除缓冲区内的数据），不会出现程序崩溃的异常情况。

另一方面，**整个程序比较冗长（大约80多行）**。虽然程序中已提供了足够多的注释信息，读者阅读和练习时可能仍然会感到难以把握。——确实，程序越长，阅读理解就越困难。

今天上机练习内容:

- 例4-8, 4-9, 4-10, 4-11, 4-12, 4-13, 4-14, 4-15, 4-16
- 练习题 4-12
- 所有程序都必须在文件头注释模块中简略地写上程序名, 并写完整学号和完整汉字姓名。最后用 SStyle 整理缩进排版格式。
- 所有文件都必须按照以前的要求命名。
- 所有文件都必须逐个上传到QQ群中。截止时间: 3月25日星期三晚上 24:00
- (如果在细节上不符合老师要求, 老师将要求罚抄相应的矫正说明 10 遍)

4.4.2 编程实例2：一个简单计算器

【例4-16】考虑一个简单的交互式计算器。假定它从键盘读入如下形式的输入行（左右两个**运算数**，中间是加减乘除**运算符**，可能夹杂有空格）：

12.8+36.5


254 - 14.38

10313 /524

程序每读入一个形式正确的表达式后就计算并输出其结果，直至用户要求结束。

【分析】很容易想到，这个程序应该有一个基本循环：

```
while (还有输入) {  
    取得数据  
    计算并输出  
}
```

$$12.8 + 36.5$$


- 程序需要依次读入左运算数、运算符和右运算数，然后计算表达式并输出。在读取数据时可以用不同的策略处理出错情况：
- 在读入左运算数时出错（遇到非数字）就直接结束程序；
- 在读入运算符时应该跳过空格以获得合法字符，然后检查如果没有获得 + - * / 这四个字符之一时就跳出循环。
- 在读入右运算数时出错则提示用户重新输入新表达式。在输入过程中用户也可以键入 EOF 来结束程序。

```

int main () {
    int op; //operator
    double left, right; //左运算数和右运算数
    cout << "小小计算器\n";
    cout << "（输入非数字字符可以退出程序）\n";
    cout << "请输入数学计算式: ";
    while ((cin >> left)) { //读入左运算数，如果读取错误则结束循环
        while ((op = cin.get()) == ' ') //跳过空格字符，直到读得运算符
            ; //空语句
        if (op != '+' && op != '-' && op != '*' && op != '/') { //如果出错
            cout << "公式错误：未读得运算符！请重新输入\n";
            cin.clear();      cin.sync();      continue; //跳出此次循环
        }
        if (!(cin >> right)) { //读入右运算数，读取错误则跳出此次循环
            cout << "公式错误：未读得右运算数！请重新输入\n";
            cin.clear();      cin.sync();      continue; //跳出此次循环
        }
    }
}

```



```

cout << "calc: " << left << " " << (char)op << " " << right << " = ";
switch(op){ // 完成计算并输出结果
    case '+': cout << left + right; break;
    case '-': cout << left - right; break;
    case '*': cout << left * right; break;
    case '/': cout << left / right; break;
    default: cout << "ERROR"; break;
}
cout << endl;
cout << "请输入数学计算式: ";
}

return 0;
}

```

读者可以多次运行这个程序，试着输入正确的表达式和错误的表达式，看程序如何处理。这个程序中是如何退出主循环？——正如前面所说，如何结束循环是一种约定。

对这一程序可以做许多改进和扩充。请读者自己考虑一些扩充并实现之。

练习题13. 改造本章正文中讨论的计算器程序，使之能处理多个连续的加减运算（例如“ $1.2 + 2.4 - 1.5 + 8.7 - 3.6 \dots$ ”）或连续的乘除运算（例如“ $3.1416 * 1.5 / 2 * 3 / 5 * 8 / 7 \dots$ ”）（为了减轻题目难度，请不必考虑加减与乘除的混和运算式）。

（提示：每次把一个运算符和它后面的运算对象视为一次循环处理的对象）

要读懂题目。

要读懂提示。

例： $1.2 + \underline{2.4} - \underline{1.5} + \underline{8.7} - \underline{3.6}$

4.4.3 编程实例3： 文件中的单词计数

现在举一个很有意思的例子，在解决这一问题的过程中，我们将采用一种有效的分析和描述问题的方法。读者不仅应该注意这个例子本身，还应当注意其中的方法。本章及后面章节的许多练习题都可能利用这种方法，它能帮助我们比较容易把问题分析清楚，把程序写得更简洁和正确。

【例4-18】 **单词计数**和**有穷自动机**的使用。一个英文纯文本文件可以看成一个字符序列。在这个序列中，有效字符被各种非打印的空白字符（包括空格' '、制表符'\t'、换行字符'\n'等）或标点符号（包括','、'.'、'\','/和';'等）以及括号（在此统称为“**分隔字符**”）分隔为一个个**单词**（为了简化起见，把数字也认为是单词）。请写一个程序统计英文纯文本文件中单词的个数，最后给出统计结果。

【分析】显然程序里需要一个计数器变量，处理中每遇到一个词时就将计数器加1。主要部分的框架可写为：

while (文件未结束)

用输入文件流的get()函数读入。容易判断。

遇到一个单词时计数器加1;

打印统计信息;

难！重点考虑！

若读的字符是单词首字符，则计数器加一。

读入字符过程中需要区分是否分隔字符。

问题：非分隔字符未必是词的开始，是否新词要看前一字符是否分隔字符。可见：不能孤立地处理，要参考前面情况。必须做情况记录，以便后面参考。

通过分析可以总结出读入过程中的前后关系：

(1) 当前字符是分隔字符，这时应该通告后面步骤：如果遇到非分隔字符，就遇到了新单词；

(2) 当前字符不是分隔字符，说明此时正处在一个单词的处理过程中，后面无论再遇到什么都不需要计数。

可看作处理过程的不同状态。两种状态：1) 读在词外（遇到非空白是新词）；2) 读在词内。

在读入字符的过程中读入状态也不断转换。

典型问题！可以用有限状态转换系统（自动机）描述。

IN/OUT（词里/词外）表示两种读入状态。

读字符的动态过程可以用下图形象描述：

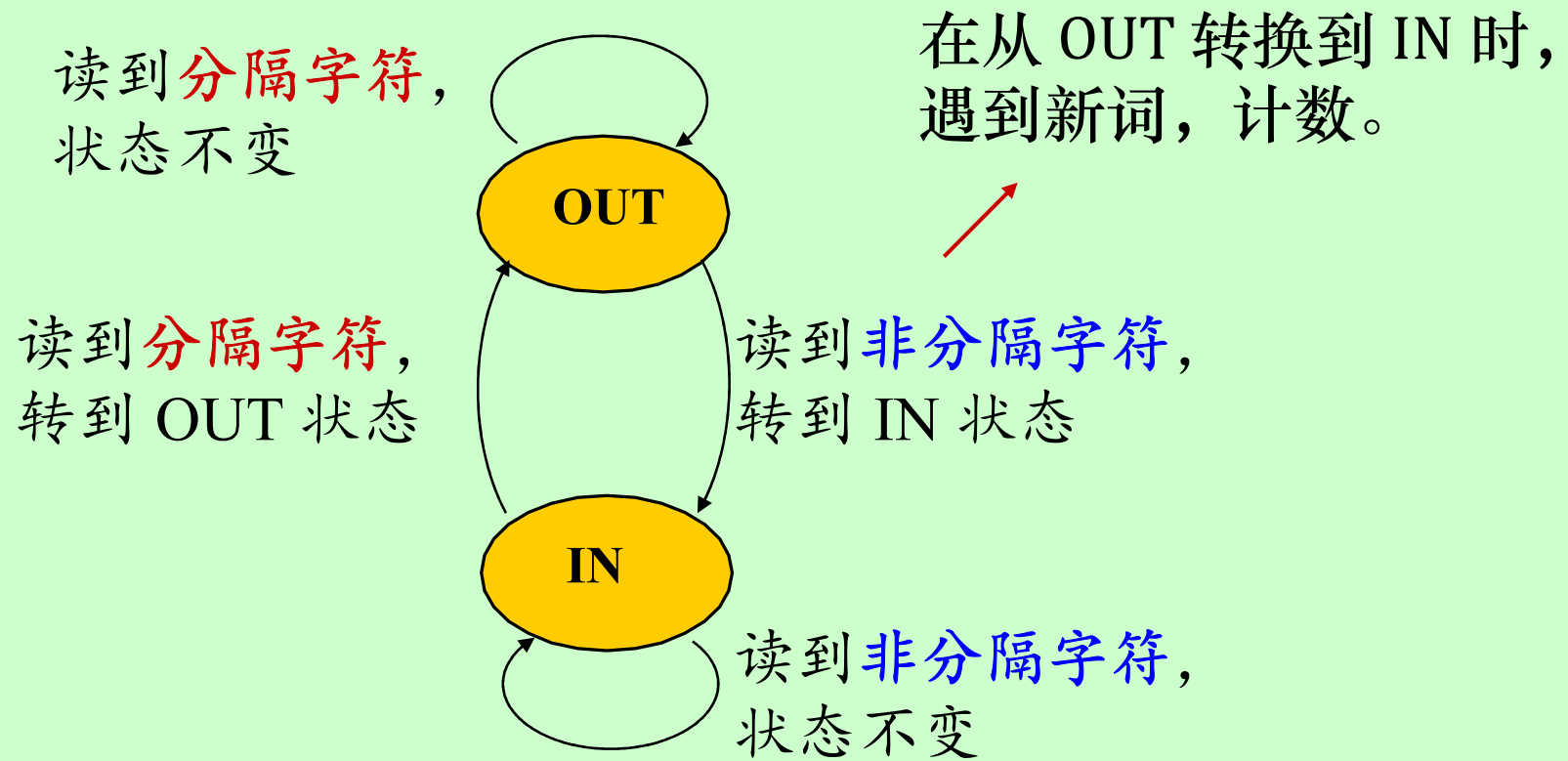


图 4.4 描述读入状态转换的自动机

对当前字符可以用标准库函数isalnum() 来判断是否为英文字母或数字字符（相应地也可判断是否分隔字符）

if (!isalnum(ch)) //ch是分隔字符（不是字母或数字字符）

if (status == IN)

status = OUT;

else //status为OUT

status = OUT;

else // ch不是分隔字符

if (status == IN)

status = IN;

else { //status为OUT

status = IN;

++counter;

}

可以稍作简化

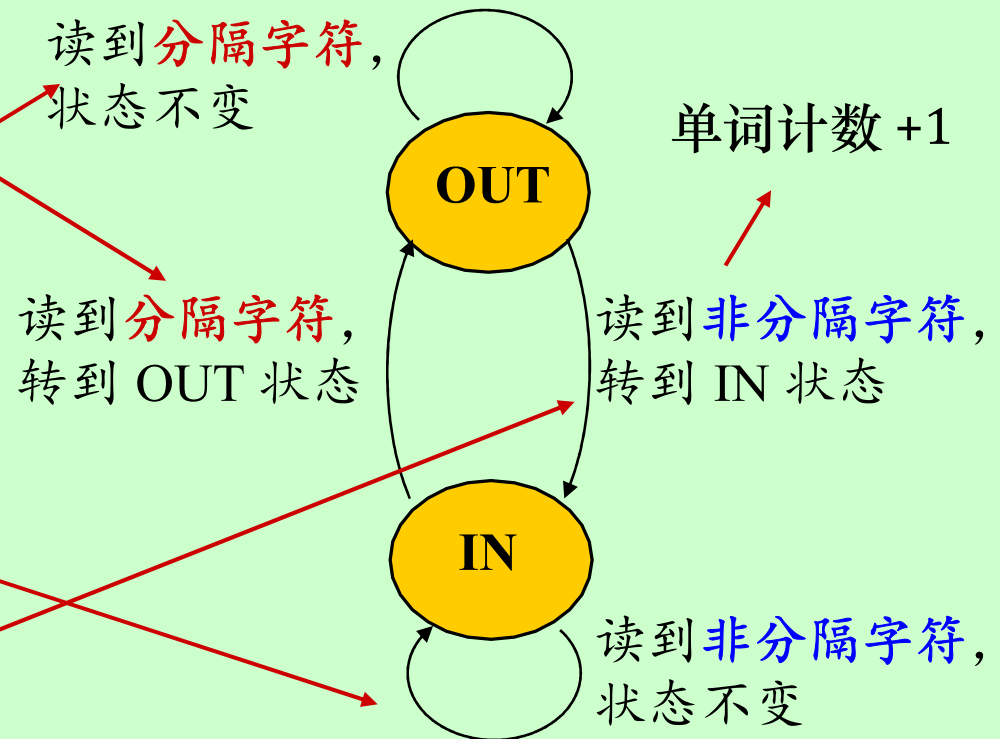


图 4.4 描述读入状态转换的自动机

```
if (!isalnum(ch)) //ch是分隔字符（不是字母或数字字符）
```

```
    if (status == IN)
        status = OUT;
```

```
else // ch不是分隔字符
```

```
    if (status != IN)
```

```
    { //status为OUT
```

```
        status = IN;
```

```
        ++counter;
```

```
    }
```

可以稍作简化

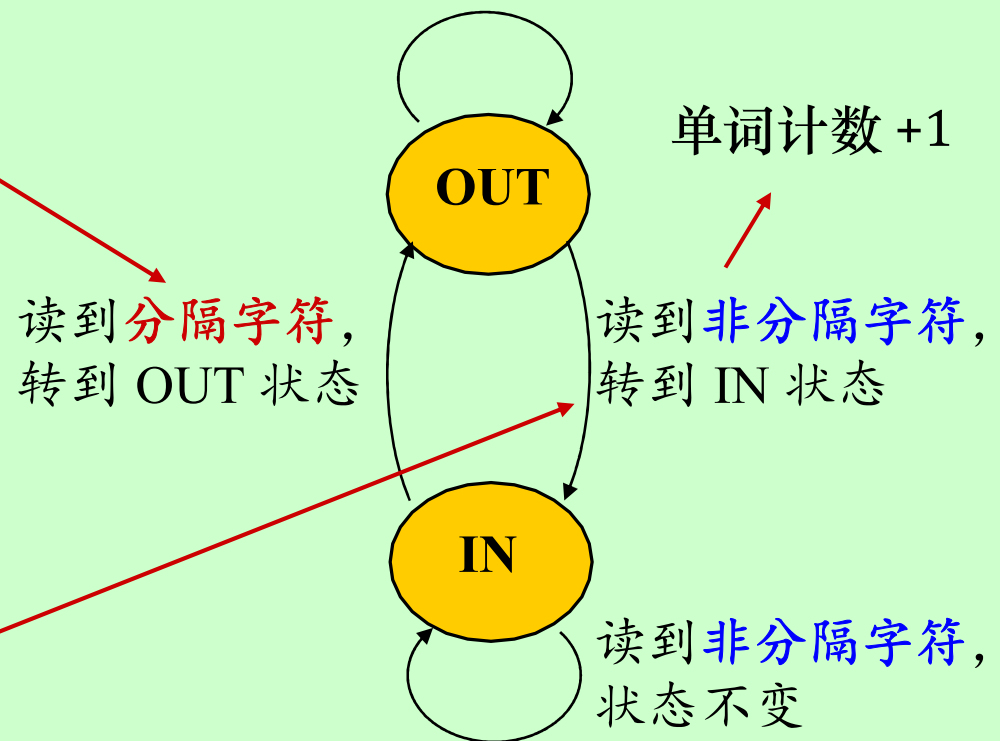


图 4.4 描述读入状态转换的自动机


```
#include <iostream>
#include <fstream>
using namespace std;

int main () {
    const int OUT = 0, IN = 1;
    int ch, counter = 0, status = OUT;
    ifstream infile("plain.txt"); //定义文件输入流并关联到文件
    if (infile)
        cout << "Reading from file: " << endl;
    else {
        cout << "ERROR: can't open input file." << endl;
        exit(1);
    }
}
```

while (文件未结束)



while ((ch = infile.get()) != EOF) { //从文件输入流中读取字符

cout.put(ch);

if (!isalnum(ch)) //ch是分隔符 (不是字母或数字字符)

status = OUT;

else if (status == OUT) { // ch不是分隔符且当前状态为OUT

status = IN;

++counter;

}

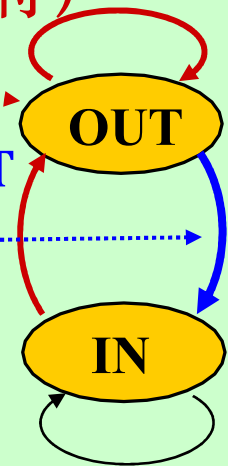
}

cout << "\ntotal words: " << counter << endl;

return 0;

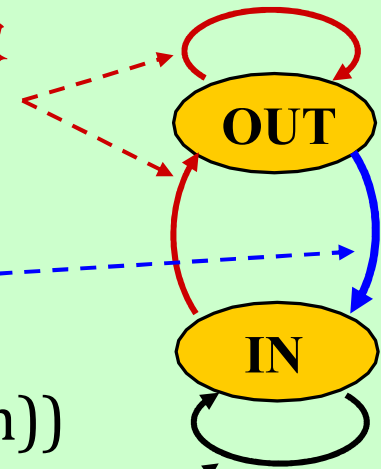
}

稍作简化的自动机



这一自动机模型的另一种实现方式：

```
int ch = ' ', counter = 0;    //给 ch 赋初值空格，保证循环正常开始
while (ch != EOF) {
    while ((ch = infile.get()) != EOF && !isalnum(ch))
        //从文件中读取字符，当ch是分隔字符时重复读取
        ; //空循环体
    if (ch == EOF) break;
    ++counter;
    while ((ch=getchar()) != EOF && isalnum(ch))
        //从文件中读取字符，当ch是字母或数字字符重复读取
        ; //空循环体
}
```



这个例子中所使用的分析方法和描述方法都很重要。

自动机是一类抽象计算模型，这里讨论的是最简单的**有限状态自动机**。

自动机理论是计算机科学的一个重要领域，既有理论价值也有实用价值。这方面的进一步情况可以参考其它书籍。人们在计算机应用和程序设计实践中还提出了许多模型。这些模型一方面能作为工具，应用于实践，作为分析问题和描述问题的工具。

另外，许多模型还有深刻的理论价值，能帮助我们进一步认识计算过程、程序、程序设计的规律性。在对计算机领域知识的进一步学习中，希望读者能有意识地关注这些模型和理论的作用及其重要性。

本书前三章中特别强调了“源程序中只能在字符串和注释中使用汉字”，而且本章所介绍的字符相关函数和本节的这个单词计数程序都只处理英文字符，没有涉及到中文字符。原因在于，对中文字符的处理有诸多复杂之处。

首先，中文字符是双字节字符（每个字符用两个字节存储表示），用char类型来处理并不完善。

第二，中文是方块字，字词之间并不使用空格作为分隔符，因此对中文进行分词远比英文要复杂。

*4.4.4 编程实例4： 图形界面程序

- 前面的程序运行时都是**字符终端界面**。也可以编写具有**图形界面**的程序。
- 需要学习一个能支持图形操作的**扩展函数库**，在程序中利用其中的函数编写图形界面。
- 推荐使用 **EGE** (Easy Graphics Engine) 图形函数库（主页：<https://xege.org/>）：可在 Windows 系统下使用的面向C/C++新手的简易图形函数库。
- EGE 图形函数库大约包含20多个与绘图相关的函数以及其它函数，从其帮助文档中可以找到所有相关信息。（本书中不作详细介绍）

【编程练习题14】 现有一个含有注释的源程序文件“test.cpp”，其主函数如下：

```
int main() { //test program
    cout << "Hello, world!" <<endl;  /* output */
    cout << "// He said:\"this is not a comment.\" \n";
    cout << '\"' << "/* That's OK. */" << '\"' << "\n";
    return 0;
}
```

写一个程序读入该源程序文件，删除该源程序中的注释，并输出保存为“testout.cpp”。（提示：1、不仅要考虑“/*”、“*/”和“//”构成的注释，而且要考虑不要误判含有仿如注释的字符串或字符。2、分析问题，可以发现文件中可以划分出四种状态：普通源代码、块注释、行注释和字符串；3、画出状态转换图；4、编程实现状态转换时，注意有多处需要用连续的两个字符进行判断。）

解答在另一个ppt文件中

- 要使用EGE 图形函数库，程序中需要写：

`#include <graphics.h>`

- 要在程序做各种绘图操作，首先需要**初始化图形环境**，做好图形窗口的初始化，使该图形窗口能接受绘图操作。
- 图形窗口定义了一套坐标，其左上角第1行第1列的绘图坐标为(0, 0)，横向的水平方向（向右）为绘图的X坐标正方向，垂直向下方向为绘图的Y坐标正方向。
- 下面来看一个简单的绘图程序实例，以了解编写图形界面程序的基本方法。

【例4-19】 使用EGE库提供的随机数生成函数和绘图函数，在一个图形窗口里连续不断地绘制线条和圆，直到用户按任意键时结束。

根据题意，使用EGE库中的函数编写程序如下：

```
#include <iostream>
#include <graphics.h>  //图形函数库
using namespace std;

int main() {
    int x, y, r;
    const int WIDTH = 640, HEIGHT = 480;
    initgraph(WIDTH, HEIGHT);  //初始化图形环境
    //使用EGE库的 outtextxy 函数在屏幕左上角输出提示性文字
    outtextxy(0, 0, "Random lines. Press any key to stop");

    randomize();  //初始化EGE库的随机数序列
    x = random(WIDTH);  //用EGE库的随机数函数random生成坐标点
    y = random(HEIGHT);
    moveto(x, y);  //使用EGE库中的 moveto 函数移动当前点到 (x, y)
```

```
do {  
    //用EGE库的 random 函数生成红绿蓝颜色分量，用 EGERGB 函数合成颜色  
    //用EGE库的 setcolor 函数设置绘图颜色  
    setcolor(EGERGB(random(256), random(256), random(256)));  
    setlinestyle(SOLID_LINE, 0, random(10) + 1, NULL); //设置线型  
    x = random(WIDTH);  
    y = random(HEIGHT);  
    lineto(x, y); //从当前点画线到(x, y)点  
    r = random(HEIGHT / 2);  
    circle(x, y, r); //以 (x, y) 点为中心，画出以 r 为半径的圆  
    Sleep(250); //暂停一小段时间  
} while (!kbhit()); //直到用户在键盘按下任意键  
  
closegraph(); //关闭图形环境  
return 0;  
}
```

- 在Dev-C++ 5.12 中文版中编写了上述程序之后，直接进行编译是通不过的，原因是默认的连接参数中缺少对相应函数库的支持。因此需要自行添加连接参数才行。操作如下：点击Dev-C++ 的菜单“工具”中的“编译选项”，在“在连接器命令行中加入以下命令”下方的文本框中添加文字。
- 如果当前使用的编译配置中包含有“32-bit”，则添加如下文字：

`-lgraphics -luuid -lmsimg32 -lgdi32 -limm32 -lole32 -loleaut32`

- 如果当前使用的编译配置中包含有“64-bit”，则添加如下文字：

`-lgraphics64 -luuid -lmsimg32 -lgdi32 -limm32 -lole32 -loleaut32`

- 只有这样设置好编译选项，才能正常地编译、连接和运行使用了EGE图形函数库的程序。

本章讨论的重要概念

- 循环程序设计，循环控制变量，累积变量，递推变量，浮点计算的误差，迭代公式（递推公式）；
- 库函数，程序计时，随机数，交互式程序，输入循环，结束标志，输入函数的返回值，文件结束与标准常量EOF，字符串类输入输出，流式文件输入输出，字符输入输出，字符相关函数；
- 状态与转换，自动机；
- 程序动态除错，程序测试，调试器，断点，逐句执行，单步进入，观察变量。

